ISSN: 1692-7257 - Volumen 2 – Número 34 - 2019
Revista Colombiana de Tecnologías de Avanzada

Recibido: 15 de enero de 2019
Aceptado:16 de abril de 2019

INTRODUCCIÓN A LA EVALUACIÓN DE CAPACIDADES: UNA REVISIÓN TEÓRICA

INTRODUCTION TO CAPACITY EVALUATION: A THEORETICAL REVIEW

M. Sc. Luis Adrian Lasso Cardona*, Emerson Rincón Reyes **, German David Estrada Holguín ***

* **Universidad del Valle.** Buga**,** Colombia. Docente, Facultad de Ingeniería de Sistemas. 3116072193. luis.lasso@correounivalle.edu.co
** **Universidad del Valle.** Buga, Colombia**.** Estudiante de pregrado, Facultad de Ingeniería de Sistemas. 3216969235. emerson.rincon@correounivalle.edu.co
*** **Universidad del Valle.** Buga, Colombia. Estudiante de pregrado, Facultad de Ingeniería de Sistemas. 3186511150. german.estrada@correounivalle.edu.co
Miembros del Grupo de Investigación en Desarrollo de Sistemas de Información y Electrónica - GIDSE

Resumen: Gracias a la globalización y a la necesidad de gestionar de manera más segura y eficiente la información generada en varias áreas, a crecido la necesidad de desarrollar software en el menor costo y tiempo posible, y que cumpla con los requerimientos exigidos por los usuarios. El desarrollo de software se convierte en una tarea exigente y de alta demanda. Por ello, es importante presentar a todos los interesados en el oficio de la programación, información relevante en cuanto a los mecanismos de evaluación de capacidades en los procesos de desarrollo de software personal y en equipo, el Modelo de Madurez de Capacidad (CMM) y la Integración del Modelo de Madurez de Capacidad (CMMI) que ayudan a fomentar habilidades y hábitos que permitan construir productos de software con altos estándares de calidad.

Palabras clave: Modelos de gestión, calidad de software, evaluación de capacidades.

Abstract: Thanks to globalization and the need to manage in a more secure and efficient way the information generated in several areas, the need to develop software at the lowest possible cost and time and that meets the requirements demanded by users has grown. Software development becomes a demanding and high demand task. For this reason, it is important to present to all those interested in the trade of programming, relevant information regarding the mechanisms for evaluating capacities in the personal and team software development processes, the Capacity Maturity Model (CMM) and the Integration of the Capacity Maturity Model (CMMI) that help to promote skills and habits that allow building software products with high quality standards.

Keywords: Management models, software quality, capacity evaluation.

1. INTRODUCCIÓN

Gracias a la globalización y a la necesidad de gestionar de manera más segura y eficiente la información generada en áreas como los negocios, la política, la salud y la industria, a crecido la necesidad de desarrollar software en el menor costo y tiempo posible, y que cumpla con estándares que garanticen su calidad. El desarrollo de software se convierte en una tarea exigente y de alta demanda, donde se requieren programadores actualizados en el uso de estándares y modelos de gestión, que les permitan trabajar individualmente, y en equipos integrados dentro de un proyecto. Los principales retos actuales para la ingeniería del software es garantizar la calidad en el proceso de desarrollo, siendo el caso de los proyectos cuyos requerimientos son gobernados por los clientes, esto

debido a los cambios del entorno en general que se pueden llegar a materializar durante la ejecución del proyecto, incrementando la complejidad del desarrollo al incluir nuevas tareas no contempladas al inicio del desarrollo. Es por ello que es labor del equipo y el ingeniero de software hacer concesiones para adaptar los lineamientos y alcances del proyecto, tratar de reducir la complejidad asociada a los problemas que surjan durante el proyecto y poder enfocar los esfuerzos a los problemas esenciales o asociados a los requerimientos (planificación, disciplina y cultura) soportándose sobre un modelo de aseguramiento de calidad (Espejo, 2016).

Según cifras de CMMI Institute (CMMIinstitute.com) líder mundial en el avance de las mejores prácticas en personas, procesos y tecnología y encargado del desarrollo y actualización de CMMI, de las organizaciones, el 50% no tienen procesos estándar, activos de proceso y ayudas de trabajo, el 42% no tienen establecido un proceso de planificación estándar, y el 41% no están preparando adecuadamente a las personas para el futuro al desarrollar sus habilidades para desarrollar capacidades organizativas. De acuerdo a estas cifras que son materia de reflexión y a la necesidad de que las organizaciones y programadores tomen conciencia de la importancia del uso de buenas prácticas de gestión de proyectos y de programación, la presente investigación tiene como finalidad presentar, información introductoria relevante en cuanto a los mecanismos de evaluación de capacidades en los procesos de desarrollo de software personal y en equipo, el Modelo de Madurez de Capacidad (CMM) y la Integración del Modelo de Madurez de Capacidad (CMMI) que ayudan a fomentar habilidades y hábitos que permitan construir productos de software con altos estándares de calidad.

2. MODELO DE MADUREZ DE CAPACIDAD (CMM)

2.1 Introducción

El Modelo de madurez de capacidad (en inglés, CMM—Capability Maturity Model) original fue desarrollado en el Instituto de Ingeniería de Software (SEI) en la Universidad Carnegie Mellon en 1986 para respaldar mejoras en la confiabilidad de las organizaciones de desarrollo de software, es decir, en su capacidad para desarrollar software de calidad a tiempo y dentro del presupuesto. Más específicamente, fue "diseñado para ayudar a los desarrolladores a seleccionar estrategias de mejora de procesos al determinar la madurez de su proceso actual e identificar los problemas más críticos para mejorar la calidad y el proceso de su software".

2.2 Propósito

CMM tiene como objetivo evaluar los procesos en sus niveles de madurez e identificar los factores que una organización debe formar para establecer una cultura de excelencia en la ingeniería de software. Los modelos de CMM se generan gracias a la experiencia colectiva de los proyectos más exitosos de software. Dentro de la familia de modelos CMM, se define el modelo para el área de software, SW-CMM. En particular, CMM es un marco de trabajo que especifica guías para organizaciones de software que quieren incrementar su capacidad de procesos, considerando los siguientes puntos: identificar fortalezas y debilidades en la organización, ponderar los riesgos de seleccionar entre diferentes contratos y monitorear los mismos, entender las actividades necesarias para planear e implementar los procesos de software, y ayudar a definir e implementar procesos de software en la organización a través de una guía.

2.3 Niveles de madurez

El modelo CMM se evalúa según el área de proceso clave (en inglés, KPA—Key Process Área), donde cada organización debe incorporar procesos adecuados en cada una de las áreas establecidas. (Weitzenfeld y Guardati, 2017). Los procesos se evalúan mediante distintos niveles de madurez, descritos en general en la Tabla 1.

Tabla 1: Descripción general de los niveles de madurez

	Nivel	Características	Transición al siguiente nivel
1	Inicial	*Ad hoc*, poca formalización, herramientas aplicadas de manera informal al proceso	Iniciar una administración rigurosa del proyecto y asegurar la calidad.
2	Repetible	Se encuentra con un proceso estable con un nivel repetible de control estadístico	Establecer un grupo y una arquitectura de proceso de desarrollo de software. Introducir métodos y tecnologías de ingeniería de software
3	Definido	Se cuenta con una base para un progreso mayor y continuo.	Establecer un conjunto básico de administraciones del proceso para identificar la calidad y costo de los parámetros, y una base de datos de proceso. Juntar y mantener los datos

			del proceso. Calcular la calidad relativa de cada producto e informar a la administración.
4	Administrado	Mejoras sustanciales en la calidad, junto con medidas comprensivas del proceso.	Apoyar la recopilación automática de datos del proceso. Usar los datos para analizar y modificar el proceso.
5	Optimizado	Mejoras con base en mayor calidad y cantidad.	Continuar mejorando y optimizando el proceso.

Fuente: http://weitzenfeld.robolat.org/wp-content/uploads/2015/01/WeitzenfeldGuardatiComputacion2008.pdf

A continuación, se describe con mayor detalle cada uno de los cinco niveles de CMM.

Nivel 1: Inicial. Expresa el estado más primitivo para las organizaciones de software. Solo reconoce que la organización es capaz de producir productos de software. La organización no tiene un proceso reconocido para la producción de software y la calidad de los productos y proyectos dependen por completo de los individuos que realizan el diseño y la implementación. Por lo común, los equipos dependen de los métodos proporcionados por un integrante del grupo que toma la iniciativa acerca del proceso que debe seguirse. El éxito de un proyecto tiene poca relación con el éxito de otro a menos que sean similares y empleen ingenieros comunes. Cuando termina un proyecto, nada se registra de su costo, tiempos o calidad. El resultado es que los nuevos proyectos suelen realizarse de una manera no más competitiva que los anteriores.

Nivel 2: Repetible. Se aplica a las organizaciones capaces de rastrear sus proyectos hasta cierto grado. Mantienen registros de los costos y tiempos del proyecto. También describen la funcionalidad de cada producto por escrito. Así, es posible predecir el costo y los tiempos de proyectos muy similares que realizan el mismo equipo. Lo que se necesita para mejorar respecto al nivel 1, es la capacidad para hacer predicciones independientes de las personas específicas que forman los equipos de proyectos.

Nivel 3: Definido. Se aplica a las organizaciones que reducen la dependencia excesiva en individuos específicos mediante la documentación del proceso e imponen un estándar, Algunas organizaciones adoptan estándares existentes, como los del Instituto de Ingeniería Eléctrica y Electrónica (IEEE), mientras que otras definen los propios. En términos generales, siempre que la administración refuerce estándares profesionales coordinados y los ingenieros los implementen de manera uniforme, la organización está en el nivel 3. Esto suele requerir capacitación especial. Se permite a los equipos flexibilidad para adaptar los estándares de la organización. Son capaces de producir aplicaciones con calidad consistente y aunque existen relativamente pocas organizaciones de este tipo, todavía les falta la capacidad de predecir. Pueden hacer pronósticos solo para proyectos muy similares y los realizados en el pasado.

Nivel 4: Administrado. Se aplica a las organizaciones que pueden predecir el costo y la programación de tareas. Una manera de hacerlo es calificar las tareas y sus componentes, y medir y registrar el costo y tiempo para diseñar e implementar esas partes. Estas mediciones constituyen los datos métricos históricos que se usan para predecir el costo y los tiempos de las tareas subsecuentes. El nivel 4 casi parece ser la capacidad máxima, pero no lo es. Se sabe que la ingeniería de software cambia con rapidez. Por ejemplo, el paradigma de la orientación a objetos invadió la metodología: los conceptos de reúso y componentes nuevos tienen un impacto creciente. Las mejoras y los paradigmas del futuro son impredecibles. Así, las capacidades de una organización de nivel 4 que no cambian de manera apropiada pueden disminuir.

Nivel 5: Optimizado. En lugar de intentar predecir los cambios futuros es preferible instituir procedimientos permanentes para buscar la explotación de métodos y herramientas nuevas mejoradas. En otras palabras, su proceso (en el sentido estricto, un proceso de metas) incluye de manera sistémica la forma de evaluar el proceso mismo de la organización. Esta es una capacidad impresionante que, a principios del año 2000, pocas organizaciones poseían. Sin embargo, con el paso del tiempo, desde que se promulgo el CMM, el nivel 5 se ha convertido en algo menos que un sueño y más que una meta. (Braude, 2003)

3. INTEGRACIÓN DEL MODELO DE MADUREZ DE CAPACIDAD (CMMI)

3.1 Introducción

A mediados de la década de los 90, el SEI decide unificar los modelos de ingeniería de software CMM-SW (Capability Maturity Model for Software, también conocido como CMM), de ingeniería de sistemas (SE-CMM Systems Engineering CMM) y de desarrollo integrado de productos (IPD-CMM Integrated Product Development CMM) (Braude, 2003), embarcándose

en un esfuerzo que culmina en el año 2002 dando origen a una nueva generación llamada CMMI (Capability Maturity Model Integration), orientado a medir la mejora en los procesos de manera individual en vez de hacerlo de manera conjunta como la representación por niveles del modelo original.

3.2 Propósito

CMMI es un modelo que especifica las mejores prácticas para desarrollar y mantener el software, las cuales fueron definidas en un esfuerzo conjunto por la industria y la comunidad académica. Dichas prácticas cubren el ciclo de vida de un producto de software, desde su concepción hasta su entrega al cliente y posterior soporte. Todos los modelos CMMI que utilizan una representación por etapas, establecen niveles en su diseño y contenido (Palacios y Porcell, 2012). En la actualidad, CMMI es uno de los programas de capacitación y certificación más populares adoptados por más de 5,000 empresas de más de 70 países, incluidos Estados Unidos, Alemania, China, Italia, India, Australia, Pakistán, etc. Muchas organizaciones y empresas internacionales exigen la madurez de CMMI para otorgar contratos, ya que CMMI se asegura de que los procesos de la organización cumplan con los estándares internacionales, por lo tanto, CMMI ayuda a una organización a competir a escala global (Saeed *et al.*, 2017).

3.3 Principios

CMMI tiene dos principios básicos a saber:

- Madurez: Atributo de las organizaciones que desarrollan o mantienen los sistemas de software. En la medida que éstas llevan a cabo su trabajo siguiendo procesos, y en la que éstos se encuentren homogéneamente implantados, definidos con mayor o menor rigor; conocidos y ejecutados por todos los equipos de la empresa, y medidos y mejorados de forma constante, las organizaciones serán más o menos "maduras".
- Capacidad: Atributo de los procesos. El nivel de capacidad de un proceso indica si sólo se ejecuta, o si también se planifica, se encuentra organizativa y formalmente definido, y se mide y se mejora de forma sistemática.

3.4 Niveles de capacidad

Los 6 niveles definidos en CMMI para medir la capacidad de los procesos son:

0. Incompleto. El proceso no se realiza, o no se consiguen sus objetivos.
1. Ejecutado. El proceso se ejecuta y se logra su objetivo.
2. Gestionado. Además de ejecutarse, el proceso se planifica, se revisa y se evalúa para comprobar que cumple los requisitos.
3. Definido Además de ser un proceso "gestionado" se ajusta a la política de procesos que existe en la organización, alineada con las directivas de la empresa.
4. Cuantitativamente gestionado. Además de ser un proceso definido se controla utilizando técnicas cuantitativas.
5. Optimizado. Además de ser un proceso cuantitativamente gestionado, de forma sistemática se revisa y modifica para adaptarlo a los objetivos del negocio.

3.5 Procesos

CMMI identifica 25 áreas de procesos (22 en la versión que no integra IPD). Vistas desde la representación continua del modelo, se agrupan en 4 categorías según su finalidad: *i*) Gestión de proyectos, *ii*) Ingeniería, *iii*) Gestión de procesos y *iv*) Soporte a las otras categorías. Vistas desde la representación escalonada, se clasifican en los 5 niveles de madurez. Al nivel de madurez 2 pertenecen las áreas de proceso cuyos objetivos debe lograr la organización para alcanzarlo. La Tabla 2, muestra las 25 áreas de proceso con su correspondiente categoría y nivel de madurez. (Palacio, 2006)

Tabla 2: Áreas de proceso de CMMI

Área de proceso	Categoría	N. mad.
Análisis y resolución de problemas	Soporte	5
Gestión de la configuración	Soporte	2
Análisis y resolución de decisiones	Soporte	3
Gestión integral de proyecto	G. Proyectos	3
Gestión integral de proveedores	G. Proyectos	3
Gestión de equipos	G. Proyectos	3
Medición y análisis	Soporte	2
Entorno organizativo para integración	Soporte	3
Innovación y desarrollo	G. Procesos	5
Definición de procesos	G. Procesos	3
Procesos orientados a la organización	G. Procesos	3
Rendimiento de los procesos de la org.	G. Procesos	4
Formación	G. Procesos	3
Integración de producto	Ingeniería	3
Monitorización y control de proyecto	G. Proyecto	2
Planificación de proyecto	G. Proyecto	2
Gestión calidad procesos y productos	Soporte	2
Gestión cuantitativa de proyectos	G. Proyectos	4
Desarrollo de requisitos	Ingeniería	3
Gestión de requisitos	Ingeniería	2
Gestión de riesgos	G. Proyectos	3
Gestión y acuerdo con proveedores	G. Proyectos	2
Solución técnica	Ingeniería	3
Validación	Ingeniería	3
Verificación	Ingeniería	3

Fuente: https://www.academia.edu/29800649/Sinopsis_de_los_modelos_SW-CMM_y_CMMI

Actualmente CMMI está en su versión V2.0, especifica prácticas globales organizadas que mejoran el rendimiento empresarial, incluyendo los mayores desafíos comunes a cualquier organización como lo son: Aseguramiento de la calidad, Ingeniería y desarrollo de productos, Servicios de entrega y gestión, Selección y gestión de proveedores, Planificación y gestión de trabajo, Gestión de la resiliencia empresarial, Gestión de la fuerza laboral, Implementación de apoyo y Mantenimiento del hábito y la persistencia.
CMMI V2.0 es un conjunto de productos integrado que consta de 5 componentes que, cuando se usan juntos, proporcionan un camino claro y comprobado para lograr sus objetivos comerciales. Estos son: *1*) Entrenamiento y certificación. *2*) Método de evaluación. *3*) Modelo. *4*) Orientación de adopción. 5) Sistemas y herramientas.

4. PROCESO DE SOFTWARE PERSONAL (PSP)

4.1 Propósito

PSP es un marco de trabajo diseñado para enseñar a los ingenieros del software a hacer mejor su trabajo. Muestra cómo estimar y planificar el trabajo, cómo controlar el rendimiento frente a esos planes y cómo mejorar la calidad de los programas. Al ser usado apropiadamente, brinda datos históricos necesarios para trabajar mejor y lograr que los elementos rutinarios del trabajo sean más predecibles y eficientes. La disciplina del PSP provee un marco estructurado para desarrollar habilidades personales y métodos necesarios para forjar al profesional informático.
El trabajo del profesional informático según PSP, se puede resumir en planificar el trabajo, hacer el trabajo de acuerdo al plan y producir productos de calidad. (Weitzenfeld y Guardati, 2017)

4.2 Principios

PSP propone un proceso. Sin embargo, cada individuo debe adaptarlo a su propia situación. Las principales ideas en las que se basa PSP son:

- Cada ingeniero es esencialmente diferente; para ser más precisos, los ingenieros deben planear su trabajo y basar sus planes en sus propios datos personales.
- Para mejorar constantemente su funcionamiento, los ingenieros deben utilizar personalmente procesos bien definidos y medidos.
- Para desarrollar productos de calidad, los ingenieros deben sentirse personalmente comprometidos con la calidad de sus productos.
- Cuesta menos encontrar y arreglar errores en la etapa inicial del proyecto que encontrarlos en las etapas subsecuentes.
- Es más eficiente prevenir defectos que encontrarlos y arreglarlos.
- La manera correcta de hacer las cosas es siempre la manera más rápida y más barata de hacer un trabajo.

4.3 Procesos

PSP está compuesto por seis procesos, los cuales están relacionados entre sí y hacen parte de una serie de etapas que avanzan hacia el mejoramiento continuo del ingeniero de software. La Fig. 1 describe de manera general los procesos de PSP.

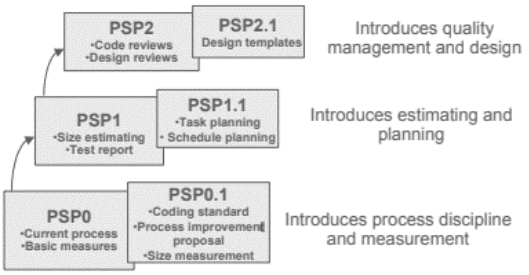

Fig. 1. Procesos del PSP
Fuente:
https://apps.dtic.mil/dtic/tr/fulltext/u2/a418430.pdf

A continuación, se describen los seis procesos de PSP:

- PSP0 y PSP0.1: Los ingenieros escriben tres asignaciones de programación usando PSP0 y PSP0.1. El objetivo es que el ingeniero aprenda cómo seguir un proceso definido y recopilar datos básicos de tamaño, tiempo y defectos.
- PSP1 y PSP1.1: Una vez que los ingenieros han recopilado algunos datos históricos, el enfoque pasa a la estimación y planificación. Los ingenieros escriben tres asignaciones de programación usando PSP1 y PSP1.1. Los ingenieros aprenden métodos estadísticos para producir estimaciones de tamaño y recursos, y usan el valor ganado para la planificación y el seguimiento del cronograma.
- PSP2 y PSP2.1: Una vez que los ingenieros tienen el control de sus planes y compromisos, el enfoque cambia a una gestión de calidad. Los ingenieros escriben cuatro asignaciones de

programación usando PSP2 y PSP2.1. Los ingenieros aprenden métodos tempranos de detección y eliminación de defectos y prácticas de diseño mejoradas.

- Informes intermedios y finales: Después de completar las primeras seis tareas, los ingenieros escriben informes de mitad de período, y una vez terminadas las diez asignaciones de programación, los ingenieros escriben los informes finales. Estos informes documentan los análisis de los ingenieros sobre su desempeño. Los ingenieros deben analizar sus datos para comprender su desempeño actual, definir objetivos desafiantes pero realistas e identificar los cambios específicos que harán para alcanzar esos objetivos. (Davis y Mullaney, 2003)

5. PROCESO DE SOFTWARE DE EQUIPO (TSP)

5.1 Propósito

TSP es un marco de trabajo que dedica atención al proceso, al producto y al trabajo en grupo. Además, basado en experiencias, guía cómo planear y administrar proyectos de software de gran tamaño que requieren ser realizados por grupos de programadores. Un equipo no se forma simplemente con la unión de varios programadores. Se necesita definir objetivos comunes, un plan de acción aceptado por todos, tener liderazgo adecuado, conocer y respetar las debilidades y fortalezas de cada miembro, ayudarse entre compañeros y ser capaz de pedir ayuda cuando se requiera. Debido a esto, resulta de fundamental importancia definir un proceso para el trabajo en equipo. Al igual que PSP, TSP requiere del registro continuo de los datos del proyecto. Los datos recopilados, a nivel personal y de equipo, sirven para hacer estimaciones más precisas de tiempo y tamaño en el futuro, así como para conocer el desempeño de los involucrados e implementar medidas para lograr mejoras. TSP provee un conjunto de formas que sirven para ese registro.

5.2 Principios

El TSP está basado en cuatro principios fundamentales:

1. El aprendizaje es más eficaz si se sigue un proceso definido y se recibe retroalimentación. TSP provee un marco de trabajo que cumple con estas características. Además, cuenta con mediciones establecidas y está diseñado para realizarse de manera cíclica. El uso de ciclos pequeños permite que el equipo reciba información sobre su desempeño periódicamente. Es decir, el trabajo del grupo se evalúa al terminar cada ciclo y los resultados se analizan y utilizan para mejorar el desempeño del mismo.
2. Para que el trabajo en equipo sea productivo, se requiere definir objetivos, un ambiente de trabajo apropiado y liderazgo adecuado.
3. Es importante recibir la guía apropiada para encontrar soluciones eficaces a los problemas de desarrollo que se originen. TSP ayuda a definir papeles, métodos y prácticas adecuadas para cada grupo de trabajo.
4. La instrucción es más eficaz cuando se construye sobre la base de conocimientos previamente adquiridos. TSP se basa en el conocimiento y las experiencias que existen sobre equipos de desarrolladores de software y material disponible sobre este tema.

5.3 Procesos

Compuesto por la siguiente estructura:

Lanzamiento: El establecimiento del equipo de trabajo se lleva a cabo en la etapa de lanzamiento. Se determinan las relaciones de trabajo, los roles de los miembros, quién va a ocupar cada uno de ellos y se llega a un acuerdo sobre los objetivos. Los roles básicos propuestos por TSPi (Team Software Process introduction, Introducción al Proceso de Software de Equipos) son: líder de equipo, administrador de desarrollo, administrador de planeación, administrador de la calidad del proceso y administrador de soporte. Los miembros del equipo a quienes se les asignan estos roles realizan también actividades propias de los ingenieros de software. Establecer objetivos es de fundamental importancia. Se requiere que los mismos estén bien definidos y sean cuantificables. Si no se pueden medir, entonces resulta difícil determinar con precisión la calidad del producto generado y del proceso seguido por el equipo. Se deben fijar objetivos ambiciosos pero realistas, de tal manera que motiven a los integrantes del grupo. Para fijar objetivos con estas características, es importante contar con experiencias previas. Cuando no se cuente con esta experiencia, TSPi sugiere adoptar, como objetivos básicos, elaborar un producto de calidad, llevar a cabo un proyecto productivo y bien administrado, y terminar el proyecto a tiempo. Cada objetivo debe acompañarse de métricas para que pueda evaluarse en qué grado fue alcanzado. Después de cada ciclo se debe evaluar el desempeño

y establecer objetivos para mejorar el proceso en el siguiente ciclo. Luego se deben definir los cambios necesarios para que los objetivos planteados puedan ser alcanzados. Además de los objetivos para el equipo, se deben establecer objetivos para cada uno de los miembros del mismo. También se deben establecer objetivos para cada uno de los roles definidos en el equipo. Es importante tener presente que, en caso de que exista un conflicto entre los objetivos del equipo y los personales o de papeles, la prioridad más alta la tiene el objetivo del equipo.

Estrategia: En esta etapa se debe idear una estrategia para realizar el trabajo, crear un diseño conceptual del producto, y hacer una estimación preliminar del tamaño del producto y del tiempo de desarrollo. El tiempo estimado debe corresponder al tiempo disponible. De no ser así, se debe revisar y ajustar la estrategia hasta que los tiempos coincidan. TSPi propone un proceso cíclico en el que cada ciclo toma la versión del sistema generada en el ciclo anterior y produce una versión aumentada. La cantidad total de ciclos depende del tamaño del sistema y del tiempo disponible. El diseño conceptual se requiere para poder hacer posteriormente la planeación del proyecto. Se debe lograr un diseño de alto nivel, definiendo la estructura general del producto. Para esto se sugiere contestar las siguientes preguntas ¿Cómo se podría desarrollar este producto?, ¿Cuáles son los principales componentes que deben construirse para lograr este producto?, ¿Cuáles son las funciones que estos componentes deben proveer? ¿Qué tan grande son estos componentes? Hecho el diseño conceptual, se puede estimar el tamaño del producto, así como el tiempo requerido para su desarrollo.

Plan: El plan indica todas las actividades y el orden en el que deben realizarse. Proporciona el marco y el contexto de trabajo. Una vez definido el plan, se puede trabajar de manera más eficiente, pues se sabe qué hacer y cuándo hacerlo. Las actividades planeadas entre los miembros del equipo deben ser distribuidas de manera equilibrada. Es decir, el trabajo asignado a cada miembro del equipo se debe poder realizar en el mismo periodo de tiempo. De esta manera no se ocasionan esperas inútiles para algunos miembros, mientras otros están saturados de trabajo. Por otra parte, los planes permiten hacer seguimientos del trabajo y avances de cada miembro del equipo. Durante el desarrollo de un producto se realizan varias tareas que se planean para ser ejecutadas en un cierto tiempo. El seguimiento permite saber en cada momento cuáles tareas fueron completadas y en qué tiempo, y de esta manera saber si se está cumpliendo con el calendario establecido. Para que los retrasos no sean excesivamente costosos, es conveniente tener planes detallados. TSPi sugiere subdividir cada actividad en unidades que no consuman más de diez horas de trabajo. De esta forma, en caso de que una tarea se retrase no implicaría más de diez horas de atraso en todo el proyecto antes de que se detecte el problema. Otro aspecto importante a tener en cuenta es que durante el desarrollo del proyecto pueden surgir algunas tareas no contempladas en el plan inicial. Por lo tanto, es conveniente incluir algunas horas en el plan para realizar esas tareas, especialmente en el primer ciclo. El resultado de este proceso es el plan completo del equipo y de cada uno de los ingenieros que lo forman, incluyendo tareas, tiempos y calidad.

Requerimientos. En esta etapa, las especificaciones de los requerimientos del sistema son generadas por el grupo de trabajo. Éstas deben ser claras y precisas. Además, el equipo debe definir indicadores que permitan evaluar el producto terminado para asegurar que éste cumpla con todas las especificaciones planteadas por el cliente. Para obtener una buena especificación de los requerimientos, normalmente se necesita dedicar tiempo junto al usuario final del sistema o un representante del mismo. Es importante que no queden dudas acerca de lo que se espera del sistema. Por lo tanto, a través de preguntas y respuestas los miembros del equipo junto al usuario deben ir definiendo y precisando todos aquellos aspectos que hayan quedado ambiguos. Algunas veces, los usuarios no tienen claro qué necesitan (o no lo saben expresar explícitamente), por lo que durante esta etapa se les debe ayudar a definir y precisar sus expectativas acerca del sistema. Una vez que se llegue a un acuerdo con el cliente sobre los requerimientos, cualquier cambio ocasionará un incremento en el costo y/o tiempo (y esto le debe quedar claro al cliente). Los requerimientos son obtenidos durante una etapa de extracción en la que se pregunta al usuario final (y/o al cliente) acerca de sus necesidades. Algunas actividades propias de la obtención de los requerimientos son: evaluar la viabilidad del sistema, entender las características de la organización, evaluar las características del negocio, identificar los usuarios del sistema, registrar las fuentes de requerimientos, definir el ambiente de operación del sistema, registrar aquellos requerimientos que fueron bien entendidos, realizar prototipos de aquellos requerimientos sobre los que se tenga alguna duda, definir posibles escenarios de uso del sistema e identificar restricciones del dominio. Durante la elaboración del documento de requerimientos, el equipo discute acerca de la necesidad del cliente, la posible

solución y cómo se va a implementar. De esta forma se genera el documento y se va logrando un acuerdo entre los miembros del grupo sobre el producto que van a desarrollar.

Diseño: En este proceso se elabora un diseño de alto nivel del sistema (es decir, de la estructura general del mismo). El diseño es un proceso creativo que permite identificar los principales componentes, así como la manera en que éstos interactúan. Esto ayuda a decidir la manera en la que se va a desarrollar cada una de las partes y cómo van a ser integradas para formar el sistema. Los estándares más importantes para realizar el diseño de alto nivel son los siguientes: *1*) Convenciones para la identificación: establecer el criterio que se va a aplicar para nombrar al sistema, sus componentes, módulos, archivos, variables y demás elementos. *2*) Formatos para la relación: especificar cómo va a ser la comunicación entre las partes del software, parámetros de entrada/salida, códigos de error u otras condiciones que deban preverse. *3*) Mensajes: establecer un estándar para escribir los mensajes de tal manera que la información proporcionada por el sistema resulte comprensible. *4*) Estándar de defectos: TSPi sugiere utilizar el estándar de defectos propuesto por PSP. *5*) Conteo de líneas de código: Si bien aún no se utilizará, TSP sugiere que en esta etapa se defina el estándar para llevar a cabo el conteo de líneas de código. *6*) Estándar para la representación del diseño: establecer un estándar para que el diseño generado pueda ser entendido sin ambigüedades por todos los miembros del equipo.

Implementación: Antes de la codificación se debe hacer un diseño más detallado, a partir del diseño de alto nivel generado en la fase anterior. Es conveniente ir subdividiendo cada parte del sistema hasta que los elementos a codificar lleguen a tener 150 o menos líneas. Una vez que se alcance este nivel de detalle, se procede a la codificación de los elementos, la cual se puede hacer de manera paralela por diferentes miembros del equipo. Aquí se define algunos estándares de implementación que aumentan y complementan los que se presentaron en la discusión sobre el proceso de diseño: *1*) Revisión de estándares: Se revisan los estándares definidos en el proceso anterior (diseño) y, si siguen resultando útiles, éstos se emplean. Se recomienda ir mejorándolos a medida que se va adquiriendo experiencia con ellos y que todos los miembros del equipo los utilicen. *2*) Estándar de codificación: Éste debe ser usado por todos los miembros, ya que esto facilitará la revisión e inspección del código, así como la medición del sistema en líneas de código, funciones, objetos o el criterio que se decida seguir. *3*) Estándar de tamaño: Permite contar líneas de código para medir el sistema. Sin embargo, durante el proyecto se generan otros productos que deben medirse y para los cuales se requiere definir nuevos estándares. Algunos ejemplos de estos productos son los documentos con los requerimientos y el diseño, pantallas y reportes, bases de datos, y reportes de pruebas. *4*) Estándar de defectos: Se sugiere adoptar el estándar propuesto en PSP. Éste es útil para poder clasificar los defectos y posteriormente analizarlos con el objetivo de mejorar el proceso. *5*) Prevención de defectos: Es importante conocer las causas que pueden ocasionar los defectos. Un ejemplo de una posible causa de defectos es la falta de conocimiento del lenguaje o del ambiente de desarrollo utilizado. Una vez detectada la causa se pueden tomar las medidas correspondientes para eliminarla.

Prueba: Es importante que la mayoría de los defectos (preferentemente todos) se hayan detectado y corregido antes de esta etapa, ya que, si no, el costo de hacerlo se incrementa notablemente. TSPi sugiere seguir las siguientes prácticas de prueba: *1*) Construir el sistema integrando unidades ya probadas. *2*) Probar el sistema integrado para evaluar si fue construido de manera correcta y verificar que están todas las partes presentes y que funcionan adecuadamente juntas. *3*) Probar el sistema para evaluar que cubre todos los requerimientos. En esta fase, mientras algunos ingenieros prueban el sistema, otros deben ir elaborando la documentación que se va a entregar al usuario. Dentro de esta documentación se deben incluir todas las explicaciones necesarias para entender la instalación, entrenamiento, uso y mantenimiento del sistema.

PostMortem: En este proceso se revisa todo el trabajo hecho por los ingenieros y todos los datos recolectados durante los procesos previos. Posteriormente se hace un análisis para identificar los puntos en los cuales se puede mejorar el proceso, lo cual provee un medio para aprender y mejorar. Se debe comparar lo planeado con lo hecho. Además, se deben detectar oportunidades para mejorar y decidir cambios en las prácticas para el siguiente ciclo o proyecto. Para determinar en qué áreas se puede mejorar el proceso se deben realizar revisiones. Algunas de las más importantes son: *1*) Revisión de calidad: Se debe comparar la calidad planeada a nivel personal y a nivel de equipo con la calidad producida. Deben plantearse las siguientes tres preguntas. ¿Qué se puede aprender de esta experiencia? ¿Qué objetivos se pueden plantear para el siguiente desarrollo? ¿Dónde se podría modificar

al proceso para mejorarlo? *2*) Evaluación de roles del equipo: Deben plantearse las siguientes tres preguntas. ¿Qué sirvió? ¿Dónde hubo problemas? ¿Qué se puede mejorar?

4. CONCLUSIONES

El PSP es uno de las metodologías de trabajo más importantes que todo programador debe conocer y comenzar a utilizar desde la fase de fundamentación, periodo en el cual se está aprendiendo a programar, para así ser más eficientes y organizados al desarrollar el código, ya que brinda las herramientas para medir el proceso y lo que es más importante, les permite formar una base para trabajar en equipo. En este aspecto, TSP se convierte en una metodología importante, exigente, y requerida para grupos de trabajo donde hay un gran número de programadores y proyectos de gran tamaño, aunque puede ser aplicado en cualquier proyecto. Esta metodología se vuelve vital para garantizar un trabajo exitoso, el cual vendrá de la mano de la capacidad de aprendizaje de los programadores y cuan familiarizados están en el proceso de software personal y de las condiciones generales del proyecto, para que este sea un éxito.

Por otro lado, muchas empresas implementan el modelo de madurez de capacidad (CMMI) con el objetivo de garantizar eficiencia en sus procesos y mayor calidad en sus productos, sin embargo, son pocas las que llegan al nivel más alto, el optimizado, pues requiere no solo un trabajo constante, requiere un alto compromiso por parte de la gerencia para establecer marcos de trabajo que exigen disciplina, entrega y participación de todos para lograrlo, lo que en muchos casos son factores que son difíciles de encontrar y relacionar, haciendo que muchas empresas desarrolladores se estanquen y no logren explotar al cien por ciento sus capacidades.

Es indudable que para lograr construir software de calidad es imperativo aplicar metodologías y marco de trabajo que guíen a directores de proyecto, analistas, diseñadores y programadores en todo el ciclo de vida del software, pero esto debe ir acompañado de responsabilidad para lograr calidad de los procesos empleados en su desarrollo y mantenimiento.

REFERENCIAS

Braude, E. J. (2003). *Ingeniería de Software una Perspectiva Orientada a Objetos*, Alfaomega. México, D.F. pp.57-60.

CMMI V2.0: Driving Performance Through Capability. https://cmmidevwebstorage.blob.core.windows.net/cmmi2landingpage/index.html (Consultado: 25 de febrero 2019)

Davis, N, y Mullaney, J. (2003), *The Team Software Process SM (TSPSM) in Practice: A Summary of Recent Results*, pp. 5-7 https://apps.dtic.mil/dtic/tr/fulltext/u2/a418430.pdf (Consultado: 28 de abril 2019)

Espejo, A. (2016). *Modelo de aseguramiento de la calidad en el proceso de desarrollo de software basado en los modelos de madurez de capacidades (CMMi), proceso de software para equipos (TSP) y personas (PSP)*, Universidad Nacional Mayor De San Marcos, Perú. http://cybertesis.unmsm.edu.pe/bitstream/handle/cybertesis/5678/Espejo_cha.pdf?sequence=1&isAllowed=y (Consultado: 8 de febrero 2019)

Palacio, J. (2006). *Sinopsis de los modelos SW-CMM y CMMI*, pp.2-3 https://www.academia.edu/29800649/Sinopsis_de_los_modelos_SW-CMM_y_CMMI (Consultado: 18 de abril 2019)

Palacios, H. y Porcell, N. (2012). *Obstáculos al implantar el modelo CMMI*. Revista EAN On-line versión ISSN 0120-8160 http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0120-81602012000100008 (Consultado: 28 de marzo 2019)

Saeed, A., Afgun, R., Akram, H., Saqlain, S. y Ghan, A. (2017). *The Impact of Capability Maturity Model Integration on Return on Investment in IT Industry: An Exploratory Case Study*. Engineering, Technology & Applied Science Research Vol. 7, No. 6. https://www.researchgate.net/publication/323550788_The_Impact_of_Capability_Maturity_Model_Integration_on_Return_on_Investment_in_IT_Industry_An_Exploratory_Case_Study (Consultado: 5 de abril 2019)

Weitzenfeld, Alfredo y Guardati, S. (2007). *Ingeniería de software: el proceso para el desarrollo de software*, Capítulo 12. pp.371-372, 375-383 y 390-391. http://weitzenfeld.robolat.org/wp-content/uploads/2015/01/WeitzenfeldGuardatiComputacion2008.pdf (Consultado: 10 de marzo 2019)